Effcient Software Model Checking
with
Block-Abstraction Memoization

Karlheinz Friedberger

LMU Munich, Germany

3. November 2021
Outline

Introduction and Background

Block-Abstraction Memoization

1. Parallel Block-Abstraction Memoization

2. Refinement for Block-Abstraction Memoization

3. Interprocedural Block-Abstraction Memoization

Conclusion
Computer’s log of Mark II Aiken Relay Calculator, Grace Hopper, 1947
(Automated) Software Verification

Computer Program

```c
int main() {
    int a = foo();
    int b = bar(a);
    if (a != b) error();
}
```

Specification

\[ \text{LTL}(G ! \text{call}(\text{error}())) \]

Verification Tool

TRUE
i.e., specification is satisfied
→ proof

FALSE
i.e., bug found
→ counterexample

Karlheinz Friedberger
LMU Munich, Germany
Fixed-point algorithm for exploring reachable abstract states

- termination based on coverage
Configurable Program Analysis (CPA)
[Beyer/Henzinger/Théoduloz, 2007]

- Fixed-point algorithm for exploring reachable abstract states
  - termination based on coverage

Operators defined for abstract domain:
- *transfer*: successor computation
- *merge*: combination of two abstract states
- *stop*: coverage of abstract states
Configurable Program Analysis (CPA)
[Beyer/Henzinger/Theodosiou, 2007]

Fixed-point algorithm for exploring reachable abstract states
▶ termination based on coverage

Operators defined for abstract domain:
▶ transfer: successor computation
▶ merge: combination of two abstract states
▶ stop: coverage of abstract states

<table>
<thead>
<tr>
<th>domain</th>
<th>abstract state</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>( l_3 )</td>
</tr>
<tr>
<td>callstack</td>
<td>([f_1, f_2])</td>
</tr>
<tr>
<td>explicit value</td>
<td>({a = 3, b = 5})</td>
</tr>
<tr>
<td>predicate</td>
<td>((l &lt; 4 \land m = 5) \lor n \neq 0)</td>
</tr>
</tbody>
</table>
Block-Abstraction Memoization
[Wonisch/Wehrheim, 2012]

Challenge:

- computation of the complete abstract state space is expensive
- analysis is not modular
Challenge:
- computation of the complete abstract state space is expensive
- analysis is not modular

Possible solution: **block summaries**
- divide and conquer strategy
- reuse intermediate results
Block-Abstraction Memoization
[Wonisch/Wehrheim, 2012]

Challenge:

- computation of the complete abstract state space is expensive
- analysis is not modular

Possible solution: **block summaries**

- divide and conquer strategy
- reuse intermediate results

Our contribution:

- independence of domain
- modular design and implementation
Block Abstraction

▶ input-output relation for a block
Block Abstraction

- input-output relation for a block

Examples for several domains:

<table>
<thead>
<tr>
<th>domain</th>
<th>abstract input state</th>
<th>abstract output state</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>$l_3$</td>
<td>$l_5$</td>
</tr>
<tr>
<td>callstack</td>
<td>$[f_1, f_2]$</td>
<td>$[f_1, f_2]$</td>
</tr>
<tr>
<td>explicit value</td>
<td>${a = 3, b = 5}$</td>
<td>${a = 4, b = 6, c = 9}$</td>
</tr>
<tr>
<td>predicate</td>
<td>$(l &lt; 4 \land m = 5) \lor n \neq 0$</td>
<td>$(l &lt; 4 \land m = 6) \lor n &gt; l + 1$</td>
</tr>
</tbody>
</table>
## Block Abstraction

▶ input-output relation for a block

Examples for several domains:

<table>
<thead>
<tr>
<th>domain</th>
<th>abstract input state</th>
<th>abstract output state</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>$l_3$</td>
<td>$l_5$</td>
</tr>
<tr>
<td>callstack</td>
<td>$[f_1, f_2]$</td>
<td>$[f_1, f_2]$</td>
</tr>
<tr>
<td>explicit value</td>
<td>${a = 3, b = 5}$</td>
<td>${a = 4, b = 6, c = 9}$</td>
</tr>
<tr>
<td>predicate</td>
<td>$(l &lt; 4 \land m = 5) \lor n \neq 0$</td>
<td>$(l &lt; 4 \land m = 6) \lor n &gt; l + 1$</td>
</tr>
</tbody>
</table>
Schematic Example of an Analysis

State-Space Exploration

A

time

Karlheinz Friedberger
LMU Munich, Germany
Schematic Example of an Analysis

State-Space Exploration
Schematic Example of an Analysis

State-Space Exploration

Block-Abstraction Memoization
Schematic Example of an Analysis

State-Space Exploration

Block-Abstraction Memoization

Parallel Block-Abstraction Memoization

Karlheinz Friedberger
LMU Munich, Germany
Parallel Block-Abstraction Memoization

[Beyer/Friedberger, 2018]

Challenges with an efficient parallel algorithm:

- program analysis strictly sequential (per block!)
- control-flow dependencies between block abstractions
Parallel Block-Abstraction Memoization

[Beyer/Friedberger, 2018]

Challenges with an efficient parallel algorithm:

- program analysis strictly sequential (per block!)
- control-flow dependencies between block abstractions

Our contribution: Parallel Block-Abstraction Memoization

- combination of the existing CPA concept and a parallel application
Evaluation for Parallel Block-Abstraction Memoization

Research questions

1. 1 processing unit: overhead of the parallel approach
2. Performance with more processing units
Evaluation for Parallel Block-Abstraction Memoization

Research questions

1. 1 processing unit: overhead of the parallel approach
2. Performance with more processing units

Configuration (CPAchecker r28809)

- explicit-value analysis (VA) with BAM or with ParallelBAM

Environment and tasks

- Intel Xeon E3-1230 v5 with 3.40 GHz and 4 physical cores
- Limitation of 15 GB RAM and 15 min of runtime
- 5400 tasks from SV benchmark suite
Evaluation: Sequential vs. Parallel Approach

- VA-BAM
- VA-ParallelBAM (1 thread)

Response time (s) vs. n-th fastest result (programs with proofs)

Response time (s) vs. n-th fastest result (programs with property violation)
Evaluation: More Processing Units

![Diagram showing the response time (s) for different numbers of threads. The x-axis represents the n-th fastest correct result, and the y-axis represents the response time in seconds. The legend indicates the number of threads: 1, 2, 4, and 8. The graph shows a trend where the response time increases with the number of threads.]
Evaluation: More Processing Units

![Graph showing the relationship between the n-th fastest correct result and response time for different numbers of threads. The graph compares 1, 2, 4, and 8 threads, with the response time on the y-axis and the n-th fastest correct result on the x-axis. The data indicates an increase in response time as the number of threads increases.]

Karlheinz Friedberger
LMU Munich, Germany
13 / 32
Refinement for Block-Abstraction Memoization

Counterexample-guided Abstraction Refinement (CEGAR)

- granularity of the analysis
- domain-independent approach
Refinement for Block-Abstraction Memoization

Counterexample-guided Abstraction Refinement (CEGAR)

- granularity of the analysis
- domain-independent approach

$S_{start}$

$S_{Error}$
Refinement for Block-Abstraction Memoization

Counterexample-guided Abstraction Refinement (CEGAR)

- granularity of the analysis
- domain-independent approach
Counterexample-guided Abstraction Refinement (CEGAR)

- granularity of the analysis
- domain-independent approach
CEGAR with Lazy Refinement (with BAM)
CEGAR with Lazy Refinement (with BAM)
CEGAR with Lazy Refinement (with BAM)
CEGAR with Lazy Refinement (with BAM)
Challenges with *In-Place* Refinement

[Beyer/Friedberger, 2018]

Missing information after the refinement

? re-compute nested blocks or take partial results from cache?
Challenges with *In-Place* Refinement

[Beyer/Friedberger, 2018]

Missing information after the refinement
- ? re-compute nested blocks or take partial results from cache?

Missing information after the analysis
- X export of incomplete data (witnesses, explored state space, statistics)
- X no guarantee for progress in the analysis
Challenges with *In-Place* Refinement

[Beyer/Friedberger, 2018]

Missing information after the refinement

? re-compute nested blocks or take partial results from cache?

Missing information after the analysis

✗ export of incomplete data (witnesses, explored state space, statistics)

✗ no guarantee for progress in the analysis

Our contribution: *Copy-on-Write* refinement for BAM

➤ no deletion of computed block abstractions
Copy-on-Write Refinement Strategy
Copy-on-Write Refinement Strategy

\[ S_{\text{start},A} \]

\[ S_{\text{start},B} \]

\[ S_{\text{start},C} \]

\[ S_{\text{Error}} \]

\[ S_{\text{cut}} \]
Copy-on-Write Refinement Strategy

$s_{\text{start},A}$

$s_{\text{start},B}$

$s_{\text{start},C}$

$s_{\text{Error}}$

$s_{\text{cut}}$

**Partial copy**
Evaluation

Research questions

- different runtime for the analysis?
- different number of refinements?
Evaluation

Research questions

- different runtime for the analysis?
- different number of refinements?

Configuration (CPAchecker r29066)

- BAM with predicate analysis (PA) and BAM with explicit-value analysis (VA)
- *in-place* vs. *copy-on-write* refinement

Environment and tasks

- Intel Xeon E3-1230 v5 with 3.40 GHz and 4 physical cores
- limitation of 15 GB RAM and 15 min of runtime
- 5400 tasks from SV benchmark suite
Evaluation: Runtime of Different Refinement Approaches

![Graph showing runtime comparison]

- **BAM with PA (in-place)**
- **BAM with PA (copy-on-write)**
- **BAM with VA (in-place)**
- **BAM with VA (copy-on-write)**
Number of refinements for (1) predicate analysis and (2) explicit-value analysis
Challenges with (intraprocedural) block abstractions:

- dependent on program context
- missing support for recursive procedures
Interprocedural Block-Abstraction Memoization

[Beyer/Friedberger, 2020]

Challenges with \(\text{intra}\)procedural block abstractions:

- \(\times\) dependent on program context
- \(\times\) missing support for recursive procedures

Our contribution: \textbf{Inter}procedural Block-Abstraction Memoization

- block abstractions are (mostly) independent of the calling context
- fixed-point algorithm for soundly analyzing recursive procedures
Interprocedural Block-Abstraction Memoization
[Beyer/Friedberger, 2020]

Challenges with (intra)procedural block abstractions:
   X dependent on program context
   X missing support for recursive procedures

Our contribution: *Inter*procedural Block-Abstraction Memoization
   ▶ block abstractions are (mostly) independent of the calling context
   ▶ fixed-point algorithm for soundly analyzing recursive procedures

✓ based on Intraprocedural Block-Abstraction Memoization
Analysis of Recursive Procedures
Analysis of Recursive Procedures
Analysis of Recursive Procedures
Analysis of Recursive Procedures
Analysis of Recursive Procedures (Interprocedural BAM)
Analysis of Recursive Procedures (Interprocedural BAM)

$s_{start,main} \xrightarrow{\text{cache miss}} s_{start,rec}$
Analysis of Recursive Procedures (Interprocedural BAM)
Analysis of Recursive Procedures (Interprocedural BAM)
Analysis of Recursive Procedures (Interprocedural BAM)
Analysis of Recursive Procedures (Interprocedural BAM)
Evaluation

Research questions

❓ effectiveness and efficiency against Intra-procedural BAM
❓ effectiveness and efficiency against other state-of-the-art tools
Evaluation

Research questions

❓ effectiveness and efficiency against Intra-procedural BAM
❓ effectiveness and efficiency against other state-of-the-art tools

Tools and configurations

▶ CPAchecker v1.9 with different analyses and different domains
▶ several participants of SV-COMP’20

Environment and tasks

▶ Intel Xeon E3-1230 v5 with 3.40 GHz and 4 physical cores
▶ limitation of 15 GB RAM and 15 min of runtime
▶ >1000 non-recursive tasks from SV benchmark suite
▶ >100 recursive tasks from SV benchmark suite
Evaluation: **CPAchecker** with Different Analyses

<table>
<thead>
<tr>
<th>Value</th>
<th>Predicate</th>
<th>Interval</th>
<th>Value+Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proofs and</td>
<td>Bugs found without BAM</td>
<td>Proofs and</td>
<td>Bugs found with BAM Intra-procedural</td>
</tr>
<tr>
<td>Proofs and</td>
<td>Bugs found with BAM Inter-procedural</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation: **CPAChecker** with Different Analyses

![Graph showing correct results for different analyses](image)

- **Value**: Proofs and Bugs found without BAM
- **Predicate**: Proofs and Bugs found with BAM *Intraprocedural*
- **Interval**: Proofs and Bugs found with BAM *Interprocedural*
- **Value+Predicate**: Proofs and Bugs found with BAM
### Evaluation: CPAchecker vs. Tools of SV-COMP’20 on Recursive Tasks

<table>
<thead>
<tr>
<th>Verifier</th>
<th>CPU time (s)</th>
<th>Proofs</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBMC</td>
<td>662</td>
<td>32</td>
<td>47</td>
</tr>
<tr>
<td>CPAchecker (Value+Predicate)</td>
<td>2180</td>
<td>37</td>
<td>46</td>
</tr>
<tr>
<td>DIVINE</td>
<td>1190</td>
<td>32</td>
<td>42</td>
</tr>
<tr>
<td>ESBMC</td>
<td>941</td>
<td>33</td>
<td>47</td>
</tr>
<tr>
<td>MAP2Check</td>
<td>23600</td>
<td>34</td>
<td>37</td>
</tr>
<tr>
<td>PINAKA</td>
<td>237</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>SYMBIOTIC</td>
<td>138</td>
<td>33</td>
<td>45</td>
</tr>
<tr>
<td>UAutomizer</td>
<td>2160</td>
<td>41</td>
<td>37</td>
</tr>
<tr>
<td>VeriAbs</td>
<td>7630</td>
<td>41</td>
<td>46</td>
</tr>
</tbody>
</table>
Conclusion

Block Abstraction Memoization
▸ domain-independent approach for software verification

Parallel Block Abstraction Memoization
▸ simple and efficient approach

Refinement for Block Abstraction Memoization
▸ insights into refinement in the context of BAM

Interprocedural Block Abstraction Memoization
▸ support for recursive procedures
▸ competitive performance
Ideas and Future Research Directions

Block Abstraction Memoization
- combine with backward analysis for bi-directional state-space exploration
- more domain

Parallel Block Abstraction Memoization
- multiple processes (machines) instead of multiple threads

Interprocedural Block Abstraction Memoization
- pointer handling and heap manipulation is currently unsolved
References and Data Availability

► Domain-Independent Multi-threaded Software Model Checking
Dirk Beyer and Karlheinz Friedberger, ASE 2018
Supplement: https://www.sosy-lab.org/research/bam-parallel/

► In-Place vs. Copy-on-Write CEGAR Refinement for Block Summarization with Caching
Dirk Beyer and Karlheinz Friedberger, ISoLA 2018
Supplement: https://www.sosy-lab.org/research/bam-cow-refinement/

► Domain-Independent Interprocedural Program Analysis using Block-Abstraction Memoization
Dirk Beyer and Karlheinz Friedberger, FSE 2020
Reproduction Package: https://doi.org/10.5281/zenodo.4024268
Tools

▶ **CPAchecker** - The Configurable Software-Verification Platform
  https://cpachecker.sosy-lab.org

▶ **BenchExec** - Reliable Benchmarking and Resource Measurement
  https://github.com/sosy-lab/benchexec

▶ **JavaSMT** - Unified Java API for SMT Solvers
  https://github.com/sosy-lab/java-smt

▶ **SV-Benchmarks** - Collection of Verification Tasks
  https://github.com/sosy-lab/sv-benchmarks
Questions?
CPAchecker Framework

Source Code → Parser & CFA Builder → CEGAR Algorithm → Results

Spec → Spec CPA, Location CPA, Callstack CPA, Predicate CPA → CPA Algorithm → CPA Algorithm

CPA Checker Algorithm

Spec CPA, Location CPA, Callstack CPA, Predicate CPA

Karlheinz Friedberger
LMU Munich, Germany
BAM in CPAchecker

Source Code → Parser & CFA Builder → CPA Algorithm → Results

wait for nested analysis

Spec → Spec CPA, Location CPA, Callstack CPA, Predicate CPA

CPA Algorithm → BAM CPA

Karlheinz Friedberger
LMU Munich, Germany
Parallel BAM in CPAchecker

Source Code → Parser & CFA Builder → CEGAR Algorithm → Parallel BAM Algorithm → CPA Algorithm → Parallel BAM CPA → Spec CPA, Location CPA, Callstack CPA, Value CPA → Results

enqueue missing block abstraction

N instances
Example of an Analysis

```c
void main(void) {
    uint a = nondet();
    uint b = nondet();
    uint s = sum(a, b);
    if (s != a + b) {
        error();
    }
}

uint sum(uint n, uint m) {
    if (n == 0) {
        return m;
    } else {
        uint tmp = sum(n - 1, m + 1);
        return tmp;
    }
}
```

Karlheinz Friedberger
LMU Munich, Germany
Example of an Analysis

control-flow automaton
Example of an Analysis

control-flow automaton

Fixed-point algorithm (first iteration)

abstract reachability graph
Example of an Analysis

control-flow automaton

Fixed-point algorithm (second iteration)

abstract reachability graph